Hacking y Pentesting con Python

Módulo 5: Librerías básicas para la automatización de procesos.

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Módulo 5: Librerías básicas para la automatización de procesos.

PRESENTACIÓN Y OBJETIVOS

En esta unidad se presentarán librerías de uso común a la hora de interactuar con servicios de acceso remoto (SSH) para ejecutar diferentes tipos de labores administrativas o de post-explotación en sistemas.

Objetivos

- Entender el funcionamiento de librerías como Paramiko para establecer conexiones con servicios SSH.
- Explorar las funciones disponibles en Paramiko y cómo se pueden utilizar en procesos de post-explotación.
- Enseñar como crear herramientas básicas para atacar servicios SSH mal configurados e inseguros.

Integración Python y SSH.

SSH es un protocolo fundamental en cualquier infraestructura y por ese motivo se encuentra soportado en prácticamente todos los sistemas basados en Unix y recientemente, en sistemas Windows. Es un protocolo que se utiliza como sustituto del clásico Telnet y permite administrar de forma remota cualquier sistema que cuente con dicho servicio en ejecución.

Desde la perspectiva de un administrador de sistemas es una herramienta fundamental, pero también lo es desde la perspectiva de un atacante ya que no solo provee de acceso a un sistema comprometido, sino que además permite realizar movimientos laterales a otros sistemas en el entorno de red por medio de la creación de túneles.

En este sentido, conviene mencionar estas características en primer lugar y posteriormente, explicar cómo se trasladan al mundo de Python por medio de librerías como Paramiko.

1. Túneles SSH

Es posible que en el objetivo se encuentre activo un servicio OpenSSH, algo que puede ser muy útil para pivotar a otros sistemas accesibles. Con OpenSSH es posible crear túneles locales, remotos o dinámicos y solamente es necesario contar con una cuenta valida o clave pública en el sistema comprometido.

La creación de túneles trae varios beneficios, el primero de ellos es que enruta de forma automática el tráfico sin necesidad de establecer modificar la configuración de rutas en el sistema.

Por otro lado, el tráfico se encuentra encapsulado en la conexión SSH, por lo tanto no será posible analizar dicho tráfico por parte de un IDS u otras soluciones de seguridad perimetral.

Los túneles locales y remotos tienen un funcionamiento similar en el sentido de que permiten abrir un puerto en el cliente (túnel local) o en el servidor (túnel remoto) cuyas conexiones entrantes serán enrutadas a una dirección IP/dominio y puerto que se especifique. La declaración de un túnel SSH tiene la siguiente sintaxis:

> Túnel Local.

ssh -L <CLIENT_INTERFACE>:<LOCAL_PORT>:<DESTINATION_INTERFACE>:<DESTINATION_PORT> user@victim

> Túnel Remoto:

ssh -R <REMOTE_INTERFACE>:<REMOTE_PORT>:<DESTINATION_INTERFACE>:<DESTINATION_PORT> user@victim

El funcionamiento de ambos mecanismos es equivalente, pero tal como se ha mencionado antes la diferencia está en que en un túnel local el puerto encargado de enrutar las peticiones a un destino concreto se abre en el cliente, es decir, en la máquina del atacante y en un túnel remoto el puerto se abre en el servidor, es decir, en la máquina de la víctima.

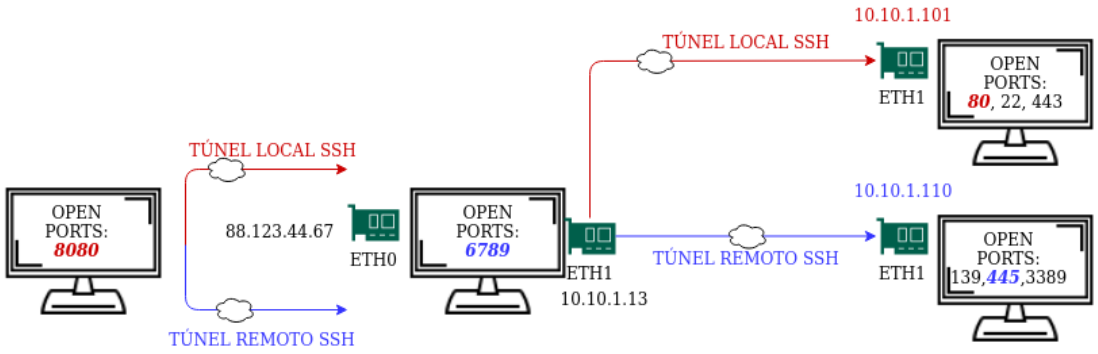
De esta manera, aunque el atacante no tenga acceso a una IP que se encuentra en un segmento de red local de la víctima, si el sistema comprometido tiene una interfaz de red que permite el acceso a dicho segmento el atacante lo utilizará como un pivote para realizar conexiones "puerto a puerto".

Se trata de un mecanismo sencillo, directo, limpio y que simplifica enormemente los movimientos laterales y posterior ataque de sistemas conectados a una red en el interior del objetivo.

No obstante, antes de poder crear un túnel local o remoto correctamente, es necesario conocer el entorno de la víctima, es decir, realizar el proceso de reconocimiento y descubrir las máquinas que se encuentran activas y sus correspondientes puertos abiertos.

Dado que los túneles con SSH permiten establecer conexiones "puerto a puerto", es necesario tener esta información previamente, descubrir la dirección IP de un sistema en el segmento de red de área local de la víctima, escanear sus puertos abiertos y seleccionar uno de ellos para crear el túnel.

La siguiente imagen enseña la diferencia entre ambos puertos con algunas direcciones IP que permitirán ilustrar los conceptos explicados anteriormente.



TÚNEL LOCAL
ssh -L 0.0.0.0:8080:10.10.1.101:80 victimuser@88.123.44.67

TÚNEL REMOTO
ssh -R 0.0.0.0:6789:10.10.1.110:445 victimuser@88.123.44.67

Figura 1.1: Túneles SSH remotos y locales

En la imagen anterior la víctima tiene dos interfaces red con una IP asignada. La IP a la que tiene acceso el atacante es la “88.123.44.67”.

En este ejemplo si se establece un túnel local se abrirá en la máquina del atacante el puerto 8080 y todas las peticiones entrantes en dicha máquina y puerto, serán automáticamente enrutadas por OpenSSH a la IP “10.10.1.101” en el puerto “80” gracias a la conexión que se ha establecido con el servidor y que evidentemente, el pivote puede conectarse a dicha IP sin ningún problema ya que cuenta con otra interfaz de red que está conectada al segmento de red “10.10.1.0/24”.

Este mismo mecanismo aplica al túnel remoto, pero en ésta ocasión el puerto que se abre es el “6789” en la máquina comprometida y todas las peticiones entrantes por dicho puerto serán enrutadas a la IP “10.10.1.110” en el puerto “445”.

Como se puede apreciar no ha sido necesario manipular rutas o ningún tipo de configuración de red en la víctima, la implementación de OpenSSH se encarga de realizar estas operaciones de forma transparente. Aunque es un buen mecanismo para pivotar, si se desea atacar a múltiples puertos de una máquina en el objetivo será necesario crear un túnel por puerto, lo que no es práctico. Por ese motivo también es posible crear túneles dinámicos.

Importante

En auditorías de seguridad, es preferible utilizar túneles locales en lugar de remotos. El motivo de esto es que la creación de túneles remotos implica una configuración adicional en el servidor que por defecto no se encuentra habilitada y en la mayoría de los casos el pentester no cuenta con los permisos necesarios para aplicar dichas configuraciones.

Un túnel dinámico hace que el pivote funcione como un proxy, es decir, que permite enrutar todas las peticiones entrantes por un puerto concreto a cualquier <IP>:<PUERTO> siempre y cuando sea accesible desde el servidor SSH (es decir, el pivote). La sintaxis para abrir un túnel dinámico es más simple y se indica con la opción "-D"

> Túnel Dinámico:

ssh -D <CLIENT_INTERFACE>:<LOCAL_PORT>user@victim

A efectos prácticos, un túnel dinámico funciona como un servidor proxy en el lado del cliente, en donde todo el tráfico se gestiona de forma trasparente gracias a la conexión SSH establecida con el servidor. Representan una alternativa útil y conveniente a los túneles locales y remotos, ya que no hace falta crear un túnel por cada puerto al que se quiera acceder en el destino.

2. Uso de Paramiko

Paramiko tiene las mismas funcionalidades que otros clientes SSH como Putty, pero evidentemente, tiene la ventaja de ser una librería que puede ser incluida en cualquier script en Python. Soporta SSH2 y otras propiedades que son habituales en OpenSSH, como por ejemplo la redirección de puertos, transferencia de ficheros segura utilizando SFTP y el establecimiento de túneles dinámicos o locales.

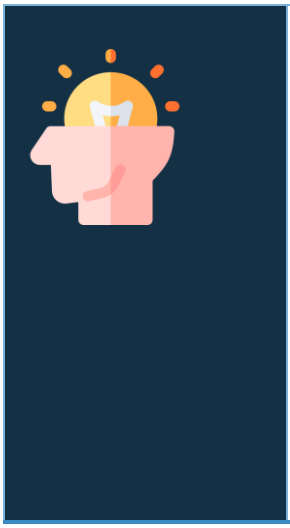
Por otro lado, para implementar los detalles de autenticación por clave pública o por contraseña, utiliza la librería PyCrypto, la cual implementa algoritmos asimétricos como RSA y DSA. Finalmente, Paramiko puede ser descargada desde la siguiente ruta: http://www.lag.net/paramiko/ para instalarla, es necesario cumplir con las dependencias especificadas en el proyecto y ejecutar el script setup.py con el argumento install.

También es posible instalar la librería usando PIP. El script **5-paramiko1.py** enseña un ejemplo simple de conexión con un servidor SSH y posterior ejecución de un comando.

En primer lugar es necesario crear una instancia de la clase SSHClient, la cual contiene todas las funciones y atributos necesarios para establecer conexiones con servidores SSH y ejecutar comandos.

La función set_missing_host_key_policy permite establecer la clave o fingerprint del servidor al que el cliente se intenta conectar. Como ocurre con cualquier cliente SSH, si la clave enviada por el servidor no es reconocida por el cliente, asume que se trata de un servidor no confiable y en el caso del cliente de OpenSSH, se enseña un mensaje indicando al usuario que para poder continuar se debe aceptar la identidad del servidor como fiable.

En el caso de Paramiko, si la firma devuelta por el servidor no es reconocida, la conexión se interrumpe inmediatamente. Si la función set_missing_host_key_policy recibe como argumento el valor devuelto por la función paramiko.AutoAddPolicy, la librería se encargará de aceptar automáticamente la firma enviada por cualquier servidor SSH.



No te olvides…

Antes de crear un cliente con Paramiko es fundamental definir la "política de host key" la cual indica cuáles son los servidores SSH confiables para establecer la conexión. En el caso de no indicar dicha política se producirá un error a la hora de crear el cliente SSH. Para realizar pruebas de pentesting en un entorno controlado, basta con indicar el valor "paramiko.AutoAddPolicy".

Por otro lado, la función exec_command se encarga de ejecutar el comando especificado contra el servidor SSH y devolver los flujos de entrada, salida y error correspondientes a la respuesta del servidor.

Finalmente, la función close se encarga de cerrar una conexión abierta con el servidor SSH. Tal como se ha mencionado anteriormente, Paramiko también soporta autenticación por clave pública contra un servidor SSH, tal como se enseña en el script **5-paramiko2.py l**a autenticación por clave pública se lleva a cabo utilizando el argumento "pkey" de la función connect, además se utiliza una instancia de la clase RSAKey para especificar la ubicación de la clave pública y su correspondiente contraseña.

Las líneas de código restantes, se encargan simplemente de ejecutar un comando y enseñar por pantalla los resultados devueltos por el servidor. Paramiko también soporta SFTP para la transferencia de ficheros entre clientes y servidores, una característica que desde el punto de vista de un atacante puede ser muy útil para subir cualquier fichero malicioso a un servidor SSH comprometido, tal como se enseña en el script **5-paramiko3-py**.

Con los ejemplos anteriores es suficiente para crear un script que se encargue de ejecutar un ataque por diccionario contra un servidor SSH objetivo y posteriormente, abrir un canal con SFTP para la transferencia y ejecución de ficheros en el servidor.

Esto es precisamente lo que se hace en el script **5-paramiko4.py.** Consiste básicamente en encontrar una cuenta de usuario válida en un servidor SSH para posteriormente, subir un fichero malicioso al servidor que se encargue de ejecutar código desarrollado por el atacante.

Si se encuentra una cuenta de usuario valida se rompe con el bucle encargado de probar cada una de las entradas del diccionario y genera un canal para la transferencia de ficheros utilizando SFTP. Al ejecutar el script es necesario especificar el fichero que se desea subir al servidor SSH, el cual puede ser otro script en Python. Después de subir el fichero al servidor, el comando chmod se encarga de establecer los permisos de ejecución adecuados y el comando nohup se encarga de mantener la ejecución del comando aunque la sesión SSH finalice. Se trata de un ataque simple utilizando solamente Paramiko.

3. PUNTOS CLAVE

- SSH es un protocolo que ha venido a sustituir al clásico Telnet. Entre sus características se encuentran el cifrado de la información extremo a extermo y la posibilidad de crear túneles.
- OpenSSH es una de las implementaciones de SSH más extendidas en sistemas basados en Unix. En el caso de sistemas basados en Windows, tiene soporte nativo como en forma de "Capability" desde Windows 10.
- Paramiko es una librería que permite crear clientes y servidores SSH. Es la librería más utilizada en aquellos casos en los que es necesario realizar conexiones SSH y administrar servidores de forma programática.
- Paramiko depende directamente de PyCrypto, por lo que se hace necesario instalar dicha librería antes de pretender instalar Paramiko. No obstante, gracias a PIP es posible instalar todo lo necesario para utilizar la librería.
- Los túneles son una de las características más potentes en OpenSSH y también se encuentran disponibles en Paramiko.